

Tight tradeoffs for approximating palindromes in streams*

Paweł Gawrychowski¹ and Przemysław Uznański²

¹Institute of Informatics, University of Warsaw, Poland

²Department of Computer Science, ETH Zürich, Switzerland

Abstract

We consider computing the longest palindrome in a text of length n in the streaming model, where the characters arrive one-by-one, and we do not have random access to the input. While computing the answer exactly using sublinear memory is not possible in such a setting, one can still hope for a good approximation guarantee.

We focus on the two most natural variants, where we aim for either additive or multiplicative approximation of the length of the longest palindrome. We first show that there is no point in considering Las Vegas algorithms in such a setting, as they cannot achieve sublinear space complexity. For Monte Carlo algorithms, we provide a lower bound of $\Omega(\frac{n}{E})$ bits for approximating the answer with additive error E , and $\Omega(\frac{\log n}{\log(1+\varepsilon)})$ bits for approximating the answer with multiplicative error $(1 + \varepsilon)$ for the binary alphabet. Then, we construct a generic Monte Carlo algorithm, which by choosing the parameters appropriately achieves space complexity matching up to a logarithmic factor for both variants. This substantially improves the previous results by Berenbrink et al. (STACS 2014) and essentially settles the space complexity.

1 Introduction

A recent trend in algorithms on strings is to develop efficient algorithms in the streaming model, where characters arrive one-by-one, and we do not have random access to the input. The main goal is to minimize the space complexity, i.e., avoid storing the already seen prefix of the text explicitly. One usually allows randomization and requires that the answer should be correct with high probability. We consider computing the longest palindrome in this model, where a palindrome is a fragment which reads the same in both directions. This is one of the basic questions concerning regularities in texts and it has been extensively studied in the classical non-streaming setting, see [1, 9, 15, 17] and the references therein. The notion of palindromes, but with a slightly different meaning, is very important in computational biology, where one considers strings over $\{A, T, C, G\}$ and a palindrome is a sequence equal to its reverse complement (a reverse complement reverses the sequences and interchanges A with T and C with G); see [10] and the references therein for a discussion of their algorithmic aspects. Our results generalize to biological palindromes in a straightforward manner.

Computing the longest palindrome in the streaming model was recently considered by Berenbrink et al. [2], who developed tradeoffs between the bound on the error and the space complexity for additive and multiplicative variants of the problem, that is, for approximating the length of the longest palindrome with either additive or a multiplicative error. Their algorithms were Monte Carlo, i.e., returned the correct answer with high probability. They also proved that any Las Vegas algorithm achieving additive error E must necessarily use $\Omega(\frac{n}{E} \log |\Sigma|)$ bits of memory, which matches the space complexity of their solution up to a logarithmic factor in the $E \in [1, \sqrt{n}]$ range, but leaves at least two questions. Firstly, does the lower bound still hold for Monte Carlo algorithms? Secondly, what is the best possible space complexity when $E \in (\sqrt{n}, n]$ in the additive variant, and what about the multiplicative version? We answer all these questions.

*up to a logarithmic factor.

Related work. The most basic problem in algorithms on strings is pattern matching, where we want to detect an occurrence of a pattern in a given text. It is somewhat surprising that one can actually solve it using polylogarithmic space in the streaming model, as proved by Porat and Porat [18]. A simpler solution was later given by Ergün et al. [6], and Breslauer and Galil [3]. Similar questions studied in such setting include multiple-pattern matching [4], approximate pattern matching [5], and parametrized pattern matching [12].

Pattern matching is also very closely related to detecting periodicities, and in fact Ergün et al. [6] also developed an efficient algorithm for computing the smallest period, where p is a period of $T[1..n]$ if $T[i] = T[i + p]$ for all $i = 1, 2, \dots, n - p$. Also palindromes are closely connected to periodicities. Informally, two long palindromes occurring close to each other imply a periodicity of the underlying fragment of the text (to the best of our knowledge, this has been first explicitly stated by Apostolico et al. [1]). Similar insights have been used to partition the text into the smallest number of palindromes [7, 11] and recognizing the so-called Pal^k language [16]. At a very high level, the idea there is to consider longer and longer prefixes of the text and maintain a succinct description of all palindromic suffixes of the current prefix. Naturally, our algorithm is based on the same high-level idea, but there are multiple non-trivial technical difficulties stemming from the fact that we cannot provide random access to the already seen part of the text, so we can only approximate such information.

Model. We work in the streaming model and consider additive and multiplicative variant of the problem. The model works as follows: we are first given the length of the text n and the bound on the desired error E (in the additive variant) or ε (in the multiplicative variant), then the characters $T[1], T[2], \dots, T[n] \in \Sigma$ arrive one-by-one. In the h -th step we receive $T[h]$ and we are required to output a number ℓ , such that the length of the longest palindrome in $T[1..h]$ is either between ℓ and $\ell + E$ (in the additive variant) or between ℓ and $(1 + \varepsilon) \cdot \ell$ (in the multiplicative variant). We have $s(n)$ bits of memory available, where we can store an arbitrary data. It is important to remember that the procedure operates in steps corresponding to the characters and we cannot retrieve an already seen character unless it has been stored in memory.

Now we are interested in the possible tradeoffs between $s(n)$ and the bound on the error. We consider Las Vegas and Monte Carlo algorithms. A Las Vegas algorithm always returns a correct answer, but its memory usage $s(n)$ is a random variable. A Monte Carlo algorithm returns a correct answer with high probability, and its memory usage $s(n)$ does not depend on the random choices, where high probability means $1 - \frac{1}{n^c}$, for arbitrarily large constant c .

We assume that the memory consists of words of size $\Omega(\log \max\{n, |\Sigma|\})$ and basic operations take $\mathcal{O}(1)$ time on such words. Bounds on the space are expressed in such words unless stated otherwise.

Previous work. The longest palindrome can be found in $\mathcal{O}(n)$ time and space (cf. Manacher [17]). Berenbrink et al. [2] constructed a streaming algorithm achieving additive error E using $\mathcal{O}(\frac{n}{E})$ space and $\mathcal{O}(\frac{n^{1.5}}{E})$ total time for any $E \in [1, \sqrt{n}]$, and a streaming algorithm guaranteeing multiplicative error $(1 + \varepsilon)$ using $\mathcal{O}(\frac{\log n}{\varepsilon \log(1 + \varepsilon)})$ space and $\mathcal{O}(\frac{n \log n}{\varepsilon \log(1 + \varepsilon)})$ total time for any $\varepsilon \in (0, 1]$, both Monte Carlo. They also proved that any Las Vegas algorithm with additive error E must necessarily use $\Omega(\frac{n}{E} \log |\Sigma|)$ bits of space.

Our results. We significantly improve on the previous results as follows and essentially settle the space complexity of the problem in both variants (see Table 1 for summary).

Firstly, we prove that any Las Vegas algorithm approximating (in either variant) the length of the longest palindrome inside a text of length n over an alphabet Σ must necessarily use $\Omega(n \log |\Sigma|)$ bits of memory (see Theorem 6.3). Hence Las Vegas randomization is simply not the right model for this particular problem. Then we move to Monte Carlo algorithms, and prove the following lower bounds on their space complexity:

- $\Omega(\frac{n}{E} \log \min\{|\Sigma|, \frac{n}{E}\})$ bits to achieve additive error E with high probability if $E \in [1, 0.49n]$ (see Theorem 6.6),*

*This can be strengthened to $0.5n - \omega(\sqrt{n})$, but for the sake of clarity we prefer to state a weaker bound, here and in subsequent similar places.

Las Vegas approximation	
$\Omega(\frac{n}{E} \log \Sigma)$	$\Omega(n \log \Sigma)$
Monte Carlo additive approximation	
$\mathcal{O}(\frac{n}{E})$ space, $\mathcal{O}(\frac{n^{1.5}}{E})$ time, $E \in [1, \sqrt{n}]$	$\mathcal{O}(\frac{n}{E})$ space, $\mathcal{O}(n \log n)$ time, $E \in [1, n]$
–	$\Omega(\frac{n}{E} \log \min\{ \Sigma , \frac{n}{E}\})$ bits, $E \in [1, 0.49n]$
Monte Carlo multiplicative approximation	
$\mathcal{O}(\frac{\log n}{\varepsilon \log(1+\varepsilon)})$ space, $\mathcal{O}(\frac{n \log n}{\varepsilon \log(1+\varepsilon)})$ time, $\varepsilon \in (0, 1]$	$\mathcal{O}(\frac{\log(n\varepsilon)}{\log(1+\varepsilon)})$ space, $\mathcal{O}(n \log n)$ time, $\varepsilon \in [\frac{2}{n}, n]$
–	$\Omega(\frac{\log n}{\log(1+\varepsilon)} \log \min\{ \Sigma , \frac{\log n}{\log(1+\varepsilon)}\})$ bits, $\varepsilon \in [n^{-0.98}, n^{0.49}]$

Table 1: A comparison of previous (on the left, c.f. [2]) and our (on the right) results. Lower bounds are in bits, and upper bounds in words consisting of $\Omega(\log \max\{n, |\Sigma|\})$ bits.

- $\Omega(\frac{\log n}{\log(1+\varepsilon)} \log \min\{|\Sigma|, \frac{\log n}{\log(1+\varepsilon)}\})$ bits to achieve multiplicative error $(1 + \varepsilon)$ with high probability if $\varepsilon \in [n^{-0.98}, n^{0.49}]$ (see Theorem 6.7).[†]

Secondly, we construct a generic Monte Carlo approximation algorithm, which by adjusting the parameters appropriately matches our lower bounds up to a logarithmic multiplicative factor. In more detail, our algorithm uses $\mathcal{O}(\frac{n}{E})$ words of space for any $E \in [1, n]$ in the additive variant (see Theorem 3.2 and Theorem 4.6) and $\mathcal{O}(\frac{\log(n\varepsilon)}{\log(1+\varepsilon)})$ words of space for any $\varepsilon \in [\frac{2}{n}, n]$ in the multiplicative variant (see Theorem 3.3 and Theorem 4.6).[‡] This essentially settles the space complexity of the problem, as it can be seen that our lower and upper bounds differ by at most a logarithmic factor for any $E \in [1, 0.49n]$ and $\varepsilon \in [n^{-0.98}, n^{0.49}]$. The time complexity of our algorithm is always $\mathcal{O}(n \log n)$ (see Theorem 5.7).

Overview of the methods. As usual in the streaming model, we apply Karp-Rabin fingerprints. We store such fingerprints for some carefully chosen prefixes of the already seen part of the text. Informally, these chosen prefixes become more and more sparse as we move closer to the beginning, with the details depending on the variant. We call such fingerprints of prefixes landmarks, and formalize this notion in Section 2. Then, in Section 3, we present a generic algorithm. The idea is that for every possible palindrome center we create a separate process which maintains the corresponding palindromic radius (or, more precisely, its approximation). By adjusting the parameters of the generic algorithm we are able to guarantee good bound on the error in both variants. For $E \in \Omega(\frac{n}{\log n})$ or $\varepsilon \in \Omega(1)$ there are only few landmarks and such generic algorithm is already efficient enough when implemented naively. To implement it efficiently for smaller E or ε , we need to avoid running processes which have already found a mismatch, but maintaining such a list explicitly might take too much space. However, multiple sufficiently long palindromes appearing close to each other imply periodicity of the corresponding fragment of the text, which can be exploited to concisely describe the whole situation. This insight (dating back to Apostolico et al. [1]) allows us to approximate the information about all active processes in logarithmic space as explained in Section 4. Then in Section 5 we use it to avoid running all active processes after reading every character. Finally, in Section 6 we apply the Yao’s minimax principle to derive the lower bounds. For Las Vegas algorithms, this is straightforward, but requires more work for Monte Carlo algorithms.

Comparison with previous work. The additive approximation algorithm proposed by Berenbrink et al. [2] uses a flat structure of \sqrt{n} fingerprints (called checkpoints). The most recently seen \sqrt{n} characters are stored explicitly, and the information of all palindromes with larger radius is compressed using the

[†]Here -0.98 can be replaced by any constant larger than -1 , and $n^{0.49}$ can be strengthened to $o(\sqrt{n})$.

[‡]For small ε this is $\mathcal{O}(\frac{\log(n\varepsilon)}{\varepsilon})$, and for large ε becomes $\mathcal{O}(\frac{\log n}{\log(1+\varepsilon)})$. Note that this does not contradict the lower bound, because $\log(n\varepsilon) = \Theta(\log n)$ for $\varepsilon \in [n^{-0.98}, n^{0.49}]$.

periodicity lemma. For multiplicative approximation, a sparse structure of checkpoints is used. Our technical contribution is of several flavors. Firstly, we use a single generic construction for both variants of the problem. Secondly, in all variants we use a hierarchical structure of fingerprints, with the fingerprints becoming more and more sparse as we move closer to the beginning. This in particular allows us to avoid storing a long suffix explicitly in the additive version. Thirdly, also the periodicity compression is applied in a hierarchical manner: we maintain a partition of the text into segments with lengths exponentially increasing with the distance from the most recently seen character and compress each such segment separately. In previous work, a rigid partition into segments of length \sqrt{n} was used. Storing such rigid partition requires $\Omega(\sqrt{n})$ space, which might be too much when the allowed error is large. Working with segments of exponentially increasing lengths allows us to decrease this additional memory usage to only $\mathcal{O}(\log n)$, but also makes the details more involved.

2 Preliminaries

For a word $w \in \Sigma^*$, we denote its length by $|w|$, and its i -th letter by $w[i]$ for any $i = 1, 2, \dots, |w|$. Similarly, $w[i..j]$ denotes the fragment starting at the i -th and ending at the j -th character, and w^R denotes the reversal, i.e., $w[|w|]w[|w| - 1]..w[1]$. The period $\text{per}(w)$ of w is the smallest natural number such that $w[i] = w[i + \text{per}(w)]$ for all $i = 1, 2, \dots, |w| - \text{per}(w)$. The well-known periodicity lemma [8] states that if p and q are periods of w and $p + q \leq |w|$, then $\text{gcd}(p, q)$ is also a period of w . We focus on detecting palindromes of even length (odd palindromes can be detected with the standard trick of duplicating every letter, see [1]). The palindromic radius at c is the largest $R(c)$ such that $T[c..(c + R(c) - 1)] = T[(c - R(c))..(c - 1)]^R$. c is the center of a palindrome $T[(c - R(c))..(c + R(c) - 1)]$.

Karp-Rabin fingerprints. We use the Karp-Rabin fingerprints [14] to quickly check equality of long strings. We choose a large prime $p \geq \max(|\Sigma|, \text{poly}(n))$ and draw $x \in \mathbb{Z}_p$ uniformly at random. Then define $f_x(w[1..k]) = (w[1] + w[2]x + \dots + w[k]x^{k-1}) \bmod p$ and define fingerprint $\Phi(w)$ of a word w to consist of $|w|$, $f_x(w)$, $f_x(w^R)$, $x^{|w|}$, $x^{-|w|}$, which takes $\mathcal{O}(1)$ space if $|w| \leq n$. The following operations take $\mathcal{O}(1)$ time:

concatenation	given $\Phi(w)$ and $\Phi(v)$, find $\Phi(wv)$,
erasing a prefix	given $\Phi(wv)$ and $\Phi(v)$, find $\Phi(w)$,
erasing a suffix	given $\Phi(wv)$ and $\Phi(v)$, find $\Phi(w)$,
reversal	given $\Phi(w)$, find $\Phi(w^R)$.

The fingerprints allow us to check if two strings are the same. Formally, we assume that $|\Sigma| \leq \text{poly}(n)$. Then, to check if $u = v$ we compare $\Phi(u)$ and $\Phi(v)$. If $u = v$ then $\Phi(u) = \Phi(v)$, and if $u \neq v$ while $|u|, |v| \leq n$ then $\Phi(u) = \Phi(v)$ with probability at most $\frac{n}{\text{poly}(n)}$. The latter situation is called a false positive. Because the running time of our algorithms will be always polynomial in n , and we will be operating on strings of length at most n , by the union bound the probability of a false positive can be made $\frac{1}{n^c}$ for any c by choosing exponent in $\text{poly}(n)$ large enough. When analyzing the correctness, we assume no false positives.

Landmarks. Our algorithms stores some values $\Phi(i) = \Phi(T[1..i])$. The intuition is that after reading $T[h]$ we calculate the fingerprint of the currently seen prefix and keep it available for some time. A landmark is a position i such that $\Phi(i)$ is currently stored. If additionally $i = 2^\lambda \cdot j$ for some j , i is a landmark on level λ , and if j is odd then i is a landmark strictly on level λ . \mathcal{Y}_λ is the set of landmarks on level λ and \mathcal{Y} is the set of all landmarks. Observe that knowing $\Phi(t)$ for all $t \in \mathcal{Y}$ is enough to calculate $\Phi(T[t + 1 .. t'])$ for any $t, t' \in \mathcal{Y}$ in $\mathcal{O}(1)$ time. Technically, \mathcal{Y} depends on the current value of h , and “ t is a landmark at h ” means that $t \in \mathcal{Y}$ just after reading $T[h]$.

For each level of landmarks we fix its size b_λ . After reading $T[h]$, the λ -th level consists of the b_λ most recently seen positions of the form $2^\lambda \cdot j$, that is, $2^\lambda(\lfloor \frac{h}{2^\lambda} \rfloor - i)$ for $i = 0, 1, \dots, b_\lambda - 1$. The sizes b_λ are chosen differently depending on the desired approximation guarantee. In all versions, the last level has number $L \leq \log n$, and $b_0 = \dots = b_{L-1}$, while no restriction is put on b_L .

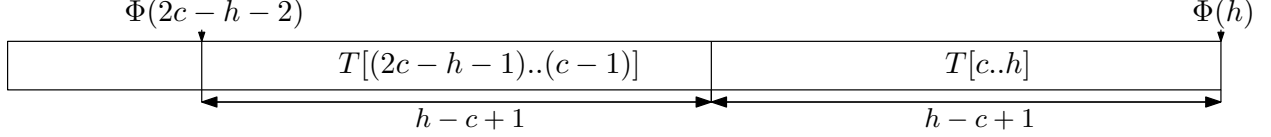


Figure 1: Checking if $R(c) \geq h - c + 1$.

For such choice of landmarks, after increasing h by one we need to add and remove at most one landmark per level, which takes $\mathcal{O}(\log h)$ time in total. The landmarks on each level are kept in a random access array with cyclic addressing, thus using $\mathcal{O}(b_\lambda + 1)$ space while allowing accesses and updates in $\mathcal{O}(1)$ time.

3 Space-efficient algorithm

We start with the basic algorithm. A proper choice of all b_λ guarantees small additive or multiplicative error, but the time and space complexity might be high. Nevertheless, the basic algorithm serves as a good starting point for developing first the space efficient version, and then finally the time efficient solution.

The idea of the algorithm is that for every possible center c we create a process $P(c)$, which keeps on computing the corresponding radius $R(c)$. We call a process alive if it has not found $T[c + \Delta]$ such that $T[c - 1 - \Delta] \neq T[c + \Delta]$ yet, and dead otherwise. The process starts with $R(c) = 0$ and then uses the landmarks to update the value of $R(c)$ (and also the final answer) whenever possible. To verify if $R(c) \geq h - c + 1$ we need to check if $T[(2c - h - 1)..h]$ is a palindrome. This requires accessing $\Phi(h)$ and $\Phi(2c - h - 2)$ to calculate $\Phi(T[(2c - h - 1)..h])$ and then $\Phi((T[(2c - h - 1)..h])^R)$, which are then compared to each other, see Fig. 1. We can simply maintain the current value of $\Phi(h)$ but retrieving $\Phi(2c - h - 2)$ is only possible when $2c - h - 2$ is a landmark. Therefore, the process $P(c)$ can update its $R(c)$ only when $2c - h - 2$ is a landmark, and doing so will be referred to as running $P(c)$ using $2c - h - 2$. If $T[(2c - h - 1)..h]$ is a palindrome, we say that $P(c)$ succeeds, and otherwise fails.

We would like to guarantee that running a process is a $\mathcal{O}(1)$ time procedure, so we need to quickly check if $2c - h - 2$ is currently a landmark (and if so, access the stored $\Phi(2c - h - 2)$). This can be easily done by iterating through all possible levels, but we want a faster method. We consider the last L -th level separately in $\mathcal{O}(1)$ time. For all lower levels, the values b_λ are all the same. We compute the largest power of 2 dividing $2c - h - 2$, call it 2^λ , then $2c - h - 2$ cannot be a landmark on level larger than λ . On the other hand, if $2c - h - 2$ is a landmark on level $\lambda' < \lambda$, then it is also a landmark on level λ . Therefore, we only need to consider the λ -th level. This allows us to run any process in $\mathcal{O}(1)$ time, and furthermore the state of any $P(c)$ can be fully described just by specifying its center c ($R(c)$ is not stored explicitly, unless mentioned otherwise). Observe that even if a process is dead, there is no harm in running it again.

The basic algorithm simply runs all processes after reading the next $T[h]$ using appropriately defined landmarks (depending on variant and desired error guarantee). For $E \in \Omega(\frac{n}{\log n})$ or $\varepsilon \in \Omega(1)$, it needs $\mathcal{O}(\log n)$ time to process $T[h]$.

Theorem 3.1. *The basic algorithm approximates the longest palindrome with additive error $E \in [1, n]$ using $\mathcal{O}(\frac{n}{E})$ time and words of memory to process $T[h]$, and the longest palindrome with multiplicative error $\varepsilon \in [\frac{2}{n}, n]$ using $\mathcal{O}(\frac{\log(n\varepsilon)}{\log(1+\varepsilon)})$ time and words of memory to process $T[h]$.*

Proof. The algorithm runs, after reading every $T[h]$, the process $P(\frac{h+y}{2})$ for every landmark y . For additive approximation, the landmarks are defined as in Theorem 3.2 with $L = \log E$. For multiplicative approximation and $\frac{2}{n} \leq \varepsilon \leq 1$, the landmarks are defined as in Theorem 3.3 with $D = \Theta(\frac{1}{\varepsilon})$, but when $1 \leq \varepsilon \leq n$ we slightly change the definition by choosing $k = \log(1 + \varepsilon)$ and keeping as a landmark, for every $i = 1, 2, \dots, \frac{\log n}{k}$, the last position divisible by $2^{k \cdot i}$. \square

An optimized version of the basic algorithm will be referred to as a scheduling scheme. A scheduling scheme should guarantee that any alive $P(c)$ such that $2c - h - 2$ is a landmark is run, unless we can either be sure that it would fail anyway, or there is another $P(c')$ such that $c' < c$ and $2c' - h - 2$ is also a landmark, and we can be sure that it succeeds (in particular, $P(c')$ is still alive).

Theorem 3.2. *Any scheduling scheme with $b_L = \infty$ approximates the longest palindrome with additive error 2^L .*

Proof. Consider an arbitrary palindrome $T[(c-x)..(c+x-1)]$. We will show that any scheduling scheme with $b_L = \infty$ returns at least $x - 2^L$. We can assume $x \geq 2^L$, then there exists $y \in (x - 2^L, x]$ such that $2^L \mid c - y - 1$. Therefore, $c - y - 1$ is permanently a landmark on level L . $P(c)$ will be alive at $c + y - 1$, so by the properties of a scheduling scheme we will run a process detecting a palindrome with radius at least $y > x - 2^L$. Thus any scheduling scheme with $b_L = \infty$ approximates the longest palindrome with additive error 2^L . \square

Theorem 3.3. *Any scheduling scheme with $b_0 = b_1 = \dots, b_{\log(n/D)} = D$ for $D \geq 6$ approximates the longest palindrome with multiplicative error $1 + \mathcal{O}(1/D)$.*

Proof. Consider an arbitrary palindrome $T[(c-x)..(c+x-1)]$. We will show that any such scheduling scheme returns at least $x/(1 + \mathcal{O}(1/D))$. Let λ be the smallest integer such that $(D-1) \cdot 2^\lambda \geq 2 \cdot x$. We have two cases.

$\lambda = 0$ After reading $T[c+x-1]$, all $c+x-1, c+x-2, \dots, c+x-D$ are landmarks on level 0, and $c-x-1$ is one of them because $2x < D$, so $T[(c-x)..(c+x-1)]$ or a longer palindrome is detected.

$\lambda > 0$ In the interval $[c-x-1, c+x-1]$ there are at most $\lfloor \frac{2x}{2^\lambda} \rfloor + 1 \leq D$ numbers divisible by 2^λ , thus there exists $y \in (x - 2^\lambda, x]$ such that $c - y - 1$ was a landmark on level λ after reading $T[c+x-1]$. As $P(c)$ is still alive at $c+x-1$, we will detect a palindrome of radius at least y . In other words, we will approximate x with additive error 2^λ . However, since λ was chosen to be minimal, we have that $2 \cdot x > (D-1) \cdot 2^{\lambda-1}$, so we can bound the multiplicative error from above by $\frac{x}{x-2^\lambda}$, which is at most $1 + \frac{1}{\Omega(D)}$ for $D \geq 6$.

Therefore, any such scheduling scheme approximates the longest palindrome with multiplicative error $1 + \mathcal{O}(1/D)$. \square

Lemma 3.4. *If $b_\lambda \geq 12$ for all $\lambda \leq L$, then for any $\Delta = 2^\ell - 1$ and for any c there is at least one $h \in [c+5\Delta, c+6\Delta]$ such that $2c - h - 2$ is a landmark at h .*

Proof. Observe that there exists unique $h \in [c+5\Delta, c+6\Delta]$ such that $2^\ell \mid 2c - h - 2$. Because $h - (2c - h - 2) \leq 2 + 12 \cdot \Delta < 12 \cdot 2^\ell$, after reading $T[h]$ there are two possibilities. If $\ell \leq L$ then $2c - h - 2$ is among the 12 last seen positions divisible by 2^ℓ . If $\ell > L$ then $2c - h - 2$ is definitely a landmark anyway. \square

4 Maintaining the alive processes

To implement a scheduling scheme, we want to know which processes are alive. Maintaining them explicitly is too space-expensive, though. Therefore, we will store a compressed approximate representation of all alive processes. Intuitively, we will group together nearby alive processes using the notion of a partition scheme described below. The representation will not be exact in the sense that it might report some dead processes as alive (but then they will have some additional properties). Such information is not providing any speedup by itself yet, but later in Section 5 we will use it to implement any scheduling scheme efficiently. In this section, we focus on maintaining the information.

Partition scheme. We maintain a partition of $T[1..h]$ into disjoint segments stored in a linked list. The length of every segment is a power of 2, their lengths are nonincreasing as one moves to the right, and there is M such that we have between A and B segments of length 2^ℓ for every $\ell = 0, 1, \dots, M-1$, and between 1 and B segments of length 2^M . A and B are constants to be specified later. After increasing h by one, a new segment of length 2^0 appears, then we possibly take two adjacent segments of length 2^ℓ such that the segment on their left (if any) is longer, and merge them into one segment of length $2^{\ell+1}$. We call this a partition scheme, as there is some flexibility as to when the merging happens.

Lemma 4.1. *There is a partition scheme with $A = 3$ and $B = 5$, which guarantees that after adding a new segment of length 2^0 we can merge in $\mathcal{O}(1)$ time at most one pair of adjacent segments of length 2^ℓ , such that there are 3 segments of the same length 2^ℓ on their right.*

Proof. This is a simple example of the recursive slow-down method of Kaplan and Tarjan [13]. Let $2^{a_1}, 2^{a_2}, 2^{a_3}, \dots$ be the lengths of the segments in the current partition, where $a_1 \leq a_2 \leq a_3 \leq \dots$. We group together all segments with the same length, and denote the number of segments of length 2^ℓ by c_ℓ . We will show how to maintain $c_\ell \in \{3, 4, 5\}$ for every $\ell = 0, 1, 2, \dots, M-1$ and $c_M \in \{1, 2, 3, 4, 5\}$, where 2^M is the maximum length of a segment in the current partition. To this end, we will keep the following invariant: if $c_i = 5$ then there exists $j \in \{0, 1, \dots, i-1\}$ such that $c_j = 3$ and $c_{j+1} = \dots = c_{i-2} = c_{i-1} = 4$. We call such partition valid.

We must show that, given a valid partition of $T[1..h]$, we can construct in $\mathcal{O}(1)$ time a valid partition of $T[1..h+1]$. We start with creating a new segment of length 2^0 and adding it to the previous partition, which increases c_0 by one. Now there are two cases.

$c_0 = 5$ We merge two (leftmost) segments of length 2^0 into a segment of length 2^1 , or in other words we decrease c_0 by two (c_0 is now equal to 3) and increase c_1 by one. Because the initial value of c_0 was 4, the only way the invariant could have been broken is that c_1 was 3, $c_2 = \dots = c_{i-1} = 4$ and $c_i = 5$ for some $i \geq 3$. But then the new c_1 becomes 4, and all c_2, c_3, \dots, c_{i-1} are now 4, so the invariant holds.

$c_0 = 4$ If there is i such that $c_1 = \dots = c_{i-2} = c_{i-1} = 4$ and $c_i = 5$, then we merge the two (leftmost) segments of length 2^i into a segment of length 2^{i+1} , which decreases c_i by two and increases c_{i+1} by one. As in the previous case, the only way the invariant could have been broken is that c_{i+1} was 3, $c_{i+2} = c_{i+3} = \dots = c_{j-1} = 4$ and $c_j = 5$ for some $j \geq i+2$. Then c_i becomes 3, and all $c_{i+1}, c_{i+2}, \dots, c_{j-1}$ are now 4, so the invariant holds.

To implement the update, we group together all consecutive i 's with the same value of c_i . In other words, we store a list of lists of segments. This allows us to find i from the second case in $\mathcal{O}(1)$ time. \square

Instead of storing every alive $P(c)$ explicitly, for every segment we group together all alive processes such that c lies inside. We need the following result, which follows from a definition of a palindrome, see [1].

Lemma 4.2. *If $c < c'$, $c' - c \leq 2^\ell$ and $R(c), R(c') \geq 2^\ell$, then $2(c' - c)$ is a period of $T[(c - 2^\ell) \dots (c' + 2^\ell - 1)]$.*

The intuition is that in a segment of length 2^ℓ either there are at most 4 alive processes which can be kept explicitly, or there are at least 5 of them and the whole segment is periodic with period at most $2^{\ell-1}$. Hence for every segment we store either a sparse or a dense description, depending (roughly) on the periodicity of the corresponding fragment.

Sparse description. We explicitly store a list of all processes inside the segment, which can be potentially still alive. We guarantee that there are at most 4 processes on that list, and that if a process is not on the list, it is surely dead. We do not guarantee that all processes on the list are still alive, but whenever we run one of them and the check fails, we declare it dead and remove from the list. The processes currently on the list are called relevant.

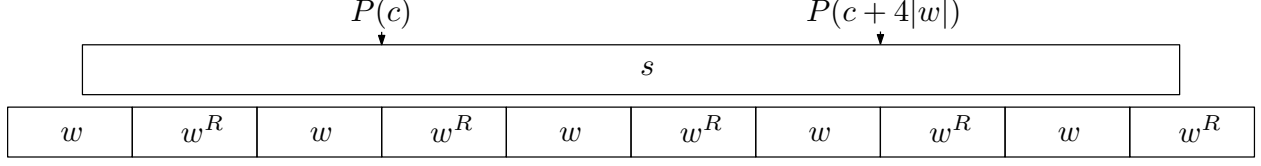


Figure 2: Alive processes inside s with a dense description are of the form $P(c + \alpha|w|)$.

Dense description. We guarantee that there exists a word w such that $|w| \leq \frac{1}{4}2^\ell$ for which the whole segment of length 2^ℓ is a subword of $(ww^R)^\infty$, see Fig. 2. Denoting the segment by s , this implies that $\text{per}(s) \leq \frac{1}{2}|s|$ and s has a palindromic subword of length $\text{per}(s)$. In such a case we store a multiple of the period, denoted by $p = k \text{per}(s) \leq \frac{1}{2}2^\ell$, such that the only alive processes inside the segment are of the form $P(c + \alpha \frac{p}{2})$ for $\alpha \geq 0$, where $T[c..(c + p - 1)]$ is an even palindrome fully within the segment. (We do not require that all such processes are still alive.) We store c and p , which is enough to run any relevant process inside s in $\mathcal{O}(1)$ time, where relevant means of the form $P(c + \alpha \frac{p}{2})$. No other process inside s can be alive.

We use Lemma 4.1 to maintain a partition of $T[1..h]$ into segments. The description of every segment requires just $\mathcal{O}(1)$ space, making the total additional space complexity $\mathcal{O}(\log n)$. After reading $T[h]$ we create a sparse description of the new segment of length 2^0 and then need to merge at most one pair of adjacent segments. After having updated the partition, we can simply run all relevant processes. Therefore, now we need to show how to merge a pair of adjacent segments s and s' of length 2^ℓ as to obtain a new segment ss' . If their descriptions are sparse, we merge the lists of s and s' and either get at most 4 processes, which constitute a valid sparse description, or at least 5 processes $P(c_1), P(c_2), \dots, P(c_5)$. The following observation follows from the Lemma 4.1.

Observation 4.3. *When a segment of length 2^ℓ is being created, the number of already seen characters on its right is at most $3 \cdot 2^{\ell-1} + 5(2^{\ell-1} - 1) = 2^{\ell+2} - 5$. When it is being destroyed, there are at least $3(2^{\ell+1} - 1)$ of them.*

Hence any $P(c_i)$ could have been run everywhere in the interval $[c_i + 2^\ell + 2^{\ell+2} - 5, c_i + 3(2^{\ell+1} - 1)]$, and by Lemma 3.4 had at least one landmark available in that interval, so its radius must be at least $2^\ell + 2^{\ell+2} - 4 \geq 2^{\ell+1}$. Therefore, we have a list of 5 processes inside a segment of length $2^{\ell+1}$, all of which have radii at least $2^{\ell+1}$ ($\ell = 0$ must be considered separately). This suffices to construct a dense description by the following lemma.

Lemma 4.4. *Given a list of $m \geq 5$ processes $P(c_1), P(c_2), \dots, P(c_m)$ inside a segment of length 2^ℓ , such that their radii are all at least 2^ℓ and no other process inside is alive, we can construct in $\mathcal{O}(m + \log n)$ time a dense description.*

Proof. We rearrange the processes so that $c_1 < c_2 < \dots < c_m$ and define $\Delta_i = c_{i+1} - c_i$. Every $2\Delta_i$ is a period of the segment by Lemma 4.2. We claim that by the periodicity lemma also $\gcd(2\Delta_1, 2\Delta_2, \dots, 2\Delta_{m-1})$ is a period of the segment. This can be seen by the following reasoning: if the radii at $c < c' < c''$ are all at least 2^ℓ , $c'' - c \leq 2^\ell$, $2d \mid 2(c' - c)$ is a period of $T[(c - 2^\ell)..(c' + 2^\ell - 1)]$ and $2d' \mid 2(c'' - c')$ is a period of $T[(c' - 2^\ell)..(c'' + 2^\ell - 1)]$, then by the periodicity lemma $2\gcd(d, d')$ is a period of the whole $T[(c - 2^\ell)..(c'' + 2^\ell - 1)]$. Then by induction $p = 2\gcd(\Delta_1, \Delta_2, \dots, \Delta_{m-1})$ is a period of $T[(c_1 - 2^\ell)..(c_m + 2^\ell - 1)]$, which contains the whole segment inside. Because $\Delta_1 + \Delta_2 + \dots + \Delta_m \leq 2^\ell$ and $m \geq 5$, $\Delta_i \leq \frac{1}{4}2^\ell$ for at least one i , so $p \leq \frac{1}{2}2^\ell$ and consequently p must be a multiple of $\text{per}(s)$.

Now we can construct a dense description. We compute p in $\mathcal{O}(m + \log n)$ with $m - 1$ applications of the Euclidean algorithm, and set $c = c_1$. Because $p \mid 2\Delta_i$ for every i , all c_i are of the form $c + \alpha \frac{p}{2}$. Furthermore, because $p \leq \min(\Delta_1, \Delta_2)$, $c + p \leq c_2$, so $T[c..(c + p - 1)]$ is fully within the segment. Finally, we must argue that $T[c..(c + p - 1)]$ is an even palindrome. First observe that $T[(c_3 - p)..(c_3 + p)]$ lies fully within the segment, and consider two cases.

- If $c_3 = c + \alpha p$, then $T[(c_3 - p)..(c_3 + p)] = T[c..(p - 1)]^2$. Because the palindromic radius at c_3 is at least $2^\ell \geq p$, $T[c..(c + p - 1)]$ is a palindrome.
- If $c_3 = c + \frac{p}{2} + \alpha p$, then $T[(c_3 - \frac{p}{2})..(c_3 + \frac{p}{2})] = T[c..(c + p - 1)]$. Because the palindromic radius at c_3 is at least $2^\ell \geq \frac{p}{2}$, $T[c..(c + p - 1)]$ is a palindrome. \square

This settles the situation when both descriptions are sparse. Before we move to the remaining case, we need an additional tool. If a description of a segment is dense, we maintain some additional information about the processes inside. Informally, we would like to know which of them are still alive, but of course we cannot afford to explicitly maintain such information. We can only afford to store a short buffer, where we keep information about a few most recently run processes. Formally, the buffer is a list of processes $P(c)$ together with their corresponding values of $R(c)$. We do not require that $P(c)$ is still alive, so it might have happened that it has been run again after reading $T[h']$ with $h < h'$, but the more recent run was unsuccessful. The buffer is updated whenever we successfully run a process $P(c)$ inside the segment. There either $P(c)$ was in the buffer, so we move it to the front and update the corresponding $R(c)$, and otherwise we prepend it to the buffer together with the current $R(c)$, and if the length of the buffer is now 6 we remove the last element from there. Hence the buffer is of length at most 5. A less trivial consequence is as follows.

Lemma 4.5. *If a segment with dense description of length 2^ℓ is being destroyed while at most 4 processes in its buffer have radii at least $2^{\ell+1}$, then no other process inside the segment can be still alive.*

Proof. By Observation 4.3 and how we process segments with dense descriptions, any $P(c)$ which might be still alive could have been run everywhere in the interval $[c + 2^\ell + 2^{\ell+2} - 5, c + 3(2^{\ell+1} - 1)]$, and by Lemma 3.4 it had at least one landmark available in that interval. Also, whenever we run any $P(c)$ inside the segment in the interval $[c + 2^\ell + 2^{\ell+2} - 5, \infty)$, and it succeeds, $R(c)$ is set to at least $2^{\ell+1}$ (except when $\ell = 0$, but then there is just one process inside the segment, so the buffer surely contains it). Therefore, if the buffer contains at most 4 processes with radii at least $2^{\ell+1}$, any $P(c)$ such that $R(c) \geq 2^{\ell+1}$ is stored in the buffer, and no other process can be still alive. \square

If at least one description is dense, by applying Lemma 4.5 to s (if its description is dense) or s' (if its description is dense), we either get that one of these segments contains at least 5 processes with radii at least $2^{\ell+1}$ in its buffer, or we get a list of at most 4 potentially still alive processes inside each segment. In the latter case we concatenate the lists to get a list of at most 8 processes inside ss' such that all other processes inside are dead. If the list contains at most 4 processes, we construct a sparse description of ss' , and otherwise we apply Lemma 4.4 to construct a dense description of ss' in $\mathcal{O}(\log n)$ time. In the former case we get a list of between 5 and 10 alive processes $P(c_1), P(c_2), \dots, P(c_m)$ inside ss' . It might be the case that there are also some other processes inside the segment which are still alive, but they are not stored in the buffer of the corresponding segment. Nevertheless, proceeding as in the proof of Lemma 4.4 we can compute in $\mathcal{O}(\log n)$ time p and c such that $T[c..(c + p - 1)]$ is an even palindrome fully within ss' , all c_i are of the form $c + \alpha \frac{p}{2}$, and $p \leq \frac{1}{2}|ss'|$ is a period of ss' . This is not a valid dense description yet, as s or s' (or both) might have dense descriptions, and we cannot guarantee that all alive processes there are of the form $c + \alpha \frac{p}{2}$.

Consider the case when s has a dense description, meaning that we have p' and c' such $T[c'..(c' + p' - 1)]$ is an even palindrome fully within s , all alive processes there are of the form $c' + \alpha \frac{p'}{2}$, and $p' \leq \frac{1}{2}|s|$ is a period of s . If $p' \mid p$ there is nothing to do. Otherwise, because the list $P(c_1), P(c_2), \dots, P(c_m)$ contains at least 5 processes inside s we have $p \leq \frac{1}{2}|s|$ and by the periodicity lemma $\gcd(p, p')$ is a period of s . Then $\gcd(p, p')$ must be actually a period of the whole ss' . Now we claim that p can be, in fact, replaced by $\gcd(p, p')$. This is because if a power of a word is a palindrome, the word itself must be a palindrome, so $T[c'..(c' + \gcd(p, p') - 1)]$ is an even palindrome.

The case when s' has a dense description, or both s and s' have dense descriptions, can be dealt with similarly.

Theorem 4.6. *For any scheduling scheme with $b_\lambda \geq 12$ for all $\lambda \leq L$, descriptions of all segments in the current partition of $T[1..h]$ can be maintained in $\mathcal{O}(\log n)$ space and $\mathcal{O}(\log n)$ time plus the time to run all relevant processes.*

5 Time-efficient algorithm

The simulation from the previous section was space-efficient, but not time-efficient yet, because there might be segments with dense descriptions and small periods, which in turn requires running many relevant processes. This is the only reason the time to process $T[h]$ might exceed $\mathcal{O}(\log n)$, as merging at most one pair of segments and running the processes in all segments with sparse descriptions takes just $\mathcal{O}(\log n)$ time. In this section we show how to simulate running all relevant processes in a segment with dense description in $\mathcal{O}(1)$ time.

Consider a dense description of a segment s . Recall that it consists of c and p , such that $T[c..(c+p-1)]$ is an even palindrome and p is a period of the whole segment, and we want to run all processes $P(c')$ inside s of the form $c' = c + \alpha \frac{p}{2}$, where $2c - h - 2$ is a landmark. We can construct and run all relevant processes in $\mathcal{O}(1)$ time each, but there might be many of them. However, there are only two consequences of running such a $P(c')$: we might update the final answer, and we might also store it in the buffer (or move it to the front there). Therefore, if we can guarantee that a particular $P(c')$ will fail anyway, we can avoid running it altogether. Similarly, if we can guarantee that many processes $P(c')$ will succeed, it is enough to run just the 5 leftmost of them. We will build on these observations to simulate running all processes of such form in a single segment with a dense description in $\mathcal{O}(1)$ total time. This is the most technical part, so we start with an overview.

Overview. We start with observing in Lemma 5.1 that, when considering such a segment, just a constant number of *associated landmark levels* needs to be considered. Then we analyze which relevant processes inside a segment should be run because of a landmark on level λ . After some basic arithmetical manipulation, we get a succinct description of all such values of c' . To avoid considering all of them, which might be too costly, we apply two lemmas characterizing the structure of palindromes in a sufficiently periodic fragment of the text, described in Lemma 5.5 and Lemma 5.6 (these observations go back to [1], but we need a slightly different formulation). To apply them, we need to compute how far the periodicity of a segment with a dense description continues to the left and to the right. To this end, we relax the notion of landmarks, introducing the so-called *ghost landmarks*, which allow us to operate on a longer suffix of the already seen $T[1..h]$. Then, using the ghost landmarks, we binary search to compute how far the periodicity extends, and apply the structural results to isolate at most 5 relevant processes, which should be run as to guarantee the correctness. To achieve the final complexity of $\mathcal{O}(\log n)$ to process $T[h]$, we precompute how far the periodicity continues when creating the segment, and then maintain this information in $\mathcal{O}(1)$ time.

Associated landmark levels. Consider a segment s . If, for some c inside s , $2c - h - 2$ is a landmark strictly on level λ at h , we say that λ is a landmark level associated to s .

Lemma 5.1. *There are at most 4 landmark levels associated to a single segment, and they can be all determined $\mathcal{O}(\log n)$ time.*

Proof. Consider a segment s of length 2^ℓ and any c inside. By Observation 4.3, when the segment is being created by merging two segments of length $2^{\ell-1}$ we have $h - c \geq 3(2^\ell - 1)$. Similarly, when the segment is being destroyed by merging with an adjacent segment of length 2^ℓ to form a segment of length $2^{\ell+1}$ we have $h - c < 2^{\ell+1} + 2^{\ell+3} - 5 = 5(2^{\ell+1} - 1)$. Consequently, we can bound $2(h - c + 1)$, which is the number of already seen characters on the right of $2c - h - 2$, as follows:

$$\begin{aligned} 2(h - c + 1) &< 10 \cdot 2^{\ell+1} - 8 \\ 2(h - c + 1) &\geq 6 \cdot 2^\ell - 4 \end{aligned}$$

If $2c - h - 2$ is a landmark strictly on level $\lambda < L$, then the number of already seen characters on its right belongs to $[b_0 \cdot 2^{\lambda-1}, b_0 \cdot 2^\lambda)$. Bounding the number of different landmark levels associated to s requires counting $\lambda < L$ such that $[b_0 \cdot 2^{\lambda-1}, b_0 \cdot 2^\lambda) \cap [6 \cdot 2^\ell - 4, 10 \cdot 2^{\ell+1} - 8) \neq \emptyset$. The condition translates into:

$$\begin{aligned} b_0 \cdot 2^{\lambda-1} &\leq 10 \cdot 2^{\ell+1} - 8 - 1 \\ b_0 \cdot 2^\lambda - 1 &\geq 6 \cdot 2^\ell - 4 \end{aligned}$$

which is equivalent to $2^\lambda \cdot b_0 \in [6 \cdot 2^\ell - 3, 40 \cdot 2^\ell - 18]$. If $\lambda = L$, the number of already seen characters on the right is at least $b_0 \cdot 2^{\lambda-1}$, so the condition becomes $2^\lambda \cdot b_0 \leq 40 \cdot 2^\ell - 18$. All in all, there are at most 4 different possible values of λ .

Generating the landmark levels associated with a given segment can be done in $\mathcal{O}(\log n)$ time by performing the above calculation. \square

Due to the above Lemma 5.1, to achieve the claimed $\mathcal{O}(\log n)$ time complexity for processing $T[h]$, we only need to show how to run all relevant processes inside a segment with a dense description using landmarks on a particular level λ associated to that segment in $\mathcal{O}(1)$ time.

Relevant processes. We need to consider all relevant processes $P(c')$, such that $2c' - h - 2$ is a landmark on level λ , implying that $2^\lambda \mid 2c' - h - 2$. The condition is equivalent to:

$$\alpha \cdot p = h + 2 - 2c \pmod{2^\lambda} \quad (1)$$

which, denoting $2^\ell = \gcd(p, 2^\lambda)$, is in turn equivalent to:

$$\alpha \cdot \frac{p}{2^\ell} = \frac{h + 2 - 2c}{2^\ell} \pmod{2^{\lambda-\ell}}$$

(unless 2^ℓ does not divide $h + 2 - 2c$, when no c' needs to be considered), so by computing the multiplicative inverse we finally get a base solution to (1):

$$\alpha_0 = \frac{h + 2 - 2c}{2^\ell} \cdot \left(\frac{p}{2^\ell}\right)^{-1} \pmod{2^{\lambda-\ell}}$$

and the general solution is:

$$\alpha = \alpha_0 + t \cdot 2^{\lambda-\ell} \quad \text{for } t \in \{\dots, -1, 0, 1, \dots\}$$

Thus we also get the solution to the original equation:

$$c' = c'_0 + t \cdot \frac{p}{2} 2^{\lambda-\ell} = c'_0 + t \cdot \frac{1}{2} \text{lcm}(2^\ell, p) \quad \text{where } c'_0 = c + \alpha_0 \frac{p}{2}. \quad (2)$$

Therefore, with a simple calculation we get a succinct description of all values of c' which should be taken into the account. Before we proceed further, let us comment on the complexity of the calculation. Since λ is fixed, both values of

$$2^\ell = \gcd(p, 2^\lambda) \quad \text{and} \quad \left(\frac{p}{2^\ell}\right)^{-1} \pmod{2^{\lambda-\ell}}$$

can be computed in $\mathcal{O}(\log n)$ time when we create the segment and stored there.

The situation now is that we have a dense description of a segment, and want to run all processes $P(c')$ of the form (2) inside the segment. Additionally, because we do not necessarily have all possible landmarks on level λ , just a few most recent, we are interested only in sufficiently large c' . Observe, that we can analyze separately processes of the following two forms:

$$P(c'_0 + t \cdot \text{lcm}(2^\lambda, p)) \quad (3)$$

$$P(c''_0 + t \cdot \text{lcm}(2^\lambda, p)) \quad \text{where } c''_0 = c'_0 + \frac{1}{2} \text{lcm}(2^\lambda, p) \quad (4)$$

From now on we will only consider the former, as the whole reasoning still holds after replacing c'_0 by c''_0 . We will also assume that only $t \geq 0$ need to be considered, which can be ensured by decreasing c'_0 by an appropriate multiple of $\text{lcm}(2^\lambda, p)$.

Because p is a period of the whole segment, $\text{lcm}(2^\lambda, p)$ is its period as well, and furthermore we can assume that $\text{lcm}(2^\lambda, p) \leq \frac{1}{2}|s|$, as otherwise there are just at most two relevant processes to run. Intuitively, knowing how far the period extends to the left and to the right allows us to restrict the number of processes to run by an argument based on the combinatorial properties of palindromes. While computing how far the period extends exactly is not possible in our setting, it can be approximated quite well using the landmarks. First, we need to introduce the notion of ghost landmarks.

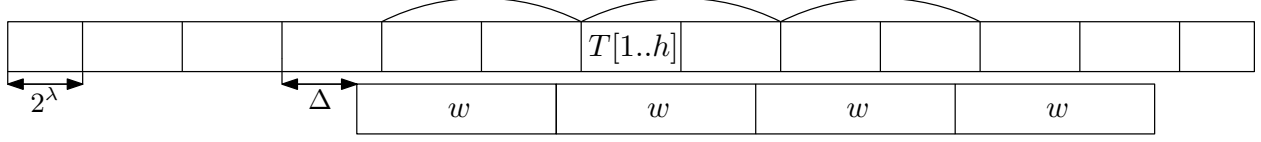


Figure 3: A number of repetitions of w such that $2^\lambda \mid |w|$ implies that $|w|$ is a period of a certain full fragment between two ghost landmarks.

Ghost landmarks. For every level of landmarks λ , we store $f_\lambda = 4 \cdot b_\lambda$ most recently seen landmarks on level λ . We call them ghost landmarks on level λ . All ghost landmarks can be maintained in the same manner as the regular landmarks, so storing them does not change the complexity of our algorithm.

Lemma 5.2. *If λ is a landmark level associated to a segment s , then for any c inside s there exists at least one ghost landmark on level λ in $T[1..(2c - h - 2)]$.*

Proof. Consider a segment s of length 2^ℓ . By Observation 4.3, the number of already seen characters on the right of s when it is being created is at least $3(2^\ell - 1)$. Let c' be any position inside s causing λ to be associated to s , i.e., $2^\lambda \mid 2c' - h' - 2$, and denote $x' = 2c' - h' - 2$. Notice that h' might be either smaller or larger than the current h . Because c' is a landmark on level λ at h' , we have that $2^\lambda \cdot b_\lambda \geq h' - x' = 2(h' - c') + 2 \geq 2 + 6(2^\ell - 1)$.

Now consider any c inside s and denote $x = 2c - h - 2$. Since c and c' both belong to the same segment of length 2^ℓ , $c' - c \leq 2^\ell - 1$. Applying Observation 4.3 again, we also get that $h - h' \leq 5(2^{\ell+1} - 1) - 1 - 3(2^\ell - 1) = 7 \cdot 2^\ell - 3$.

Thus the number of already seen characters on the right of x can be bounded as follows:

$$h - x = 2h - 2c + 2 \leq (2h' - 2c' + 2) + 2(2^\ell - 1) + 14 \cdot 2^\ell - 6.$$

Because $h' - x' = 2(h' - c') + 2 \geq 2 + 6(2^\ell - 1)$, we have $16 \cdot 2^\ell - \frac{32}{3} \leq \frac{8}{3} \cdot (h' - x')$, so the above bound can be rewritten as:

$$h - x \leq \frac{11}{3} \cdot (h' - x') + \frac{8}{3} \leq \frac{11}{3} b_\lambda \cdot 2^\lambda + \frac{8}{3} \leq (4b_\lambda - 1) \cdot 2^\lambda$$

where the last inequality holds because $b_\lambda \geq 12$. Since the leftmost ghost landmark on level λ has at least $(4b_\lambda - 1) \cdot 2^\lambda$ already seen characters on its right, by the above calculation it must be on the left of $x = 2c - h - 2$ as claimed. \square

Now going back to approximating how far the period extends to the left and to the right, we proceed as follows. We choose w of length $\text{lcm}(2^\lambda, p)$ starting at $T[c'_0]$. Because we have adjusted c'_0 so that only $t \geq 0$ need to be considered and $|w| \leq \frac{1}{2}|s|$, we can assume that w is fully within the segment. We know that the whole segment can be covered by repeating w to the left and to the right (where, possibly, the last repetition is a suffix or a prefix of w , respectively), and would like to figure out how far we can continue that until we hit either a boundary of the already seen $T[1..h]$, or a subword of length $|w|$ which is different than w . This can be approximated quite well using the ghost landmarks, if w repeats at least twice.

Lemma 5.3. *For any w such that $T[i..(i + 2|w| - 1)] = w^2$, $2^\lambda \mid |w|$, and $T[1..(i - 1)]$ contains at least one ghost landmark on level λ , we can compute in $\mathcal{O}(\log h)$ time $r \geq 2$ such that $T[i..(i + r|w| - 1)] = w^r$ and either $i + (r + 2)|w| > h$ or $T[i..(i + (r + 2)|w| - 1)] \neq w^{r+2}$.*

Proof. By the assumption about ghost landmark on level λ , we can access any $\Phi(2^\lambda \cdot j)$ with $j \geq \lfloor \frac{i-1}{2^\lambda} \rfloor$ in $\mathcal{O}(1)$ time. Hence if we are lucky and $i = 2^\lambda \cdot j + 1$, we can compute $\Phi(T[(i + \alpha|w|)..(i + \beta|w| - 1)])$ for any $0 \leq \alpha \leq \beta$ in $\mathcal{O}(1)$ time, which allows us to binary search for r in $\mathcal{O}(\log h)$ time. In more detail, to check if $T[i..(i + r|w| - 1)] = w^r$ we check if $|w|$ is a period of $T[i..(i + r|w| - 1)]$, which can be done by comparing $\Phi(T[(i + |w|)..(i + r|w| - 1)])$ and $\Phi(T[i..(i + (r - 1)|w| - 1)])$.

In the general case, let $i = 2^\lambda \cdot j + 1 + \Delta$, where $\Delta \in [0, 2^\lambda)$. If $T[i..(i + r|w| - 1)] = w^r$ and $r \geq 2$, then $|w|$ is a period of $T[(2^\lambda \cdot j + 2^\lambda + 1)..(2^\lambda \cdot j + 2^\lambda + \alpha|w|)]$, see Fig. 3, where $\alpha = r - 1$. In the other direction, if $|w|$ is a period of $T[(2^\lambda \cdot j + 2^\lambda + 1)..(2^\lambda \cdot j + 2^\lambda + \alpha|w|)]$ and $r \geq 2$, then $r \geq \alpha$. (The assumption that $r \geq 2$ is crucial.) Hence we can determine the largest α such that $|w|$ is a period of $T[(2^\lambda \cdot j + 2^\lambda + 1)..(2^\lambda \cdot j + 2^\lambda + \alpha|w|)]$ in $\mathcal{O}(\log h)$ time using ghost landmarks on level λ , and then simply return α , which guarantees $r \in \{\alpha, \alpha + 1\}$. \square

Lemma 5.4. *For any w such that $T[i..(i + 2|w| - 1)] = w^2$, $2^\lambda \mid |w|$, and $T[1..(2i - h - 2)]$ contains at least one ghost landmark on level λ , we can compute in $\mathcal{O}(\log h)$ time $\ell \geq 0$ such that $T[(i - \ell|w|)..(i - 1)] = w^\ell$ and either $i - (\ell + 2)|w| < 2i - h - 2$ or $T[(i - (\ell + 2)|w|)..(i - 1)] \neq w^{\ell+2}$.*

Proof. The proof will be very similar to the proof of Lemma 5.3, except that we have to take into the account the fact that while there might be many more repetitions of w to the left, we might not have enough ghost landmarks on level λ to detect them.

Let $i = 2^\lambda \cdot j + 1 + \Delta$, where $\Delta \in [0, 2^\lambda)$. If $T[(i - \ell|w|)..(i - 1)] = w^\ell$, then $|w|$ is a period of $T[(2^\lambda \cdot j - \alpha|w| + 2^\lambda + 1)..(2^\lambda \cdot j + |w| + 2^\lambda)]$, where $\alpha = \ell$. In the other direction, if $|w|$ is a period of $T[(2^\lambda \cdot j - \alpha|w| + 2^\lambda + 1)..(2^\lambda \cdot j + |w| + 2^\lambda)]$, then $\ell \geq \alpha - 1$. So we only need to binary search for the largest α such that $|w|$ is a period of $T[(2^\lambda \cdot j - \alpha|w| + 2^\lambda + 1)..(2^\lambda \cdot j + |w| + 2^\lambda)]$ and return $\max(0, \alpha - 1)$. The remaining difficulty is that $2^\lambda \cdot j - \alpha|w| + 2^\lambda$ might lie too far on the left to be a ghost landmark on level λ , so the binary search needs to be slightly modified. We first choose the largest α_0 such that $2^\lambda \cdot j - \alpha_0|w| + 2^\lambda$ is a ghost landmark on level λ . There are two possibilities.

1. $|w|$ is a period of $T[(2^\lambda \cdot j - \alpha_0|w| + 2^\lambda + 1)..(2^\lambda \cdot j + |w| + 2^\lambda)]$, then the largest α might exceed α_0 . But we can return $\ell = \max(0, \alpha_0 - 1)$, because then $i - (\ell + 1)|w| \leq i - \alpha_0|w|$, and the choice of α_0 and the assumption, by Lemma 5.2, implies $i - (\alpha_0 + 1)|w| < 2i - h - 2$, so $i - (\ell + 2)|w| < 2i - h - 2$.
2. Otherwise, we binary search over all $\alpha \leq \alpha_0$, and return $\max(0, \alpha - 1)$.

We can binary search for α_0 in $\mathcal{O}(\log h)$ time, so the total time is $\mathcal{O}(\log h)$. \square

We apply Lemma 5.3 and Lemma 5.4 to approximate how many times w repeats on its right and on its left in $T[(2c'_0 - h - 2)..(c'_0 - 1)]$ with accuracy 1, assuming that w^2 occurs at $T[c'_0]$. Notice that there might be many more repetitions to the left in the whole $T[1..(c'_0 - 1)]$, but Lemma 5.4 does not allow us to detect all of them. Now the crucial insight is that even though we do now know the exact number of repetitions, we can iterate through the at most 4 possible combinations of the number of repetitions to the left and to the right, and additionally consider the possibility that there is only a single occurrence of w in the segment. Hence we need to iterate through 5 possibilities in total. For each such combination, we will restrict the number of processes which should be run, therefore by running the processes determined for each of these combinations we will not lose the correctness. Hence from now on we assume that we know the exact number of repetitions of w to the left and to the right.

We need the following two simple structural results, which allow us to bound the palindromic radius in a sufficiently periodic subword of the text. A similar (in spirit) argument appeared already in [1], but we need a slightly different formulation. We say that a palindrome centered at c reaches h if $R(c) \geq h - c + 1$.

Lemma 5.5. *Consider uw^kv starting at position i in $T[1..h]$, where $|u| = |w| = |v|$, w is a palindrome, and $u, v \neq w$. For any $\alpha \in \{1, 2, \dots, k\}$, if the palindrome centered at $i + \alpha|u|$ reaches h then $\alpha = \frac{k}{2} + 1$.*

Proof. Take any $\alpha \in \{1, 2, \dots, k\}$. For a palindrome centered at $i + \alpha|u|$ to reach c , $R(i + \alpha|u|)$ must be at least $\min(\alpha - 1, k + 1 - \alpha)|u|$. But then either $u = w^R$ or $v = w^R$, which is a contradiction. \square

Lemma 5.6. *Consider uw^k starting at position i in $T[1..h]$, where $|u| = |w|$, w is a palindrome, $u \neq w$, and $h - i - (k + 1)|u| + 1 < |w|$. For any $\alpha \in \{1, 2, \dots, \lceil \frac{k}{2} \rceil\}$, the palindrome centered at $i + \alpha|u|$ cannot reach h . Additionally, either all palindromes centered at $i + \alpha|u|$ with $\alpha \in \{\lceil \frac{k}{2} \rceil + 1, \dots, k\}$ reach h , or none of them do.*

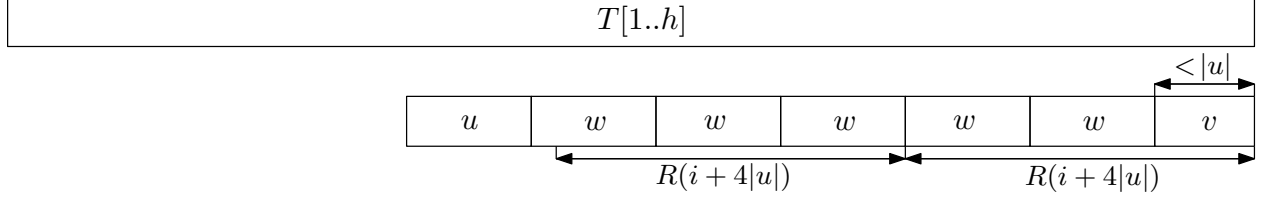


Figure 4: All palindromes centered at positions $i + \alpha|u|$ with $\alpha \in \{\lceil \frac{k}{2} \rceil + 1, \dots, k\}$ reach h if v is a prefix of w .

Proof. Take any $\alpha \in \{1, 2, \dots, k\}$. If $\alpha \leq \lceil \frac{k}{2} \rceil$, then because $u \neq w$ the radius at $i + \alpha|u|$ is too small for the palindrome centered at $i + \alpha|u|$ to reach h . Otherwise, let $T[i..h] = uw^k v$, where $|v| < |u|$ because $h - i - (k + 1)|u| + 1$, see Fig. 4. Now either v is not a prefix of w , and we actually get the situation from Lemma 5.5, so only $\alpha = \frac{k}{2} + 1$ can possibly correspond to a palindrome reaching h , or v is a prefix of w , and for all $\alpha \geq \lceil \frac{k}{2} \rceil + 1$ the palindrome centered at $i + \alpha|u|$ reaches h . \square

Recall that we want to run all $P(c'_0 + t|w|)$ inside the segment with $t \geq 0$, and our w starts at $T[c'_0]$. We know that w repeats ℓ times to the left in $T[(2c'_0 - h - 2)..(c'_0 - 1)]$ and r times to the right till the end of the already seen $T[1..h]$. The actual number of repetitions of w to the left in the whole $T[1..(c'_0 - 1)]$, denoted ℓ' , might be larger than ℓ . By Lemma 5.5 and Lemma 5.6, either all processes of the form $P(c'_0 + (-\ell' + \alpha)|w|)$ with $\alpha \geq \lceil \frac{\ell' + r}{2} \rceil + 1$ will succeed, or just the one with $\alpha = \frac{\ell' + r}{2} + 1$ will succeed. Because the size of the buffer is 5, we only need to ensure that the 5 leftmost processes which will succeed are run. To guarantee this, we run all processes of the form $P(c'_0 + (\max(-\ell + \lceil \frac{\ell' + r}{2} \rceil + 1, 0) + x)|w|)$ for $x = 0, 1, 2, 3, 4$ which are still inside the segment. This is correct, as following two cases show.

1. The process $P(c'_0 + (-\ell' + \alpha)|w|)$ with $\alpha = \frac{\ell' + r}{2} + 1$ is on the left of the segment, so either all or none processes of such form in the segment are alive.
2. The process $P(c'_0 + (-\ell' + \alpha)|w|)$ with $\alpha = \frac{\ell' + r}{2} + 1$ is inside the segment, so ℓ' cannot be too large. More precisely, $\ell' \leq r$, and consequently $\ell = \ell'$.

We run a constant number of processes, each of them in $\mathcal{O}(1)$ time, but to ensure that every segment is processed in such complexity, we also need to remove the binary search used to approximate how many times w can be repeated to the left and to the right.

Recall that $|w| = \text{lcm}(2^\lambda, p)$, w starts at $T[c'_0]$ and lies fully within a segment s , and furthermore $|w|$ is a period of the whole s . As mentioned before, we can also assume that $|w| \leq \frac{1}{2}|s|$, as otherwise there are at most two processes which might need to be run. We can compute how many times w can be repeated to its left (or rather approximate this value as described in Lemma 5.4) when the segment is created, as the result does not depend on the current value of h . Similarly, we can compute how many times it can be repeated to the right when we create the segment, but here the important difference is that we might continue till the very end of the current $T[1..h]$, i.e., the next copy of w might extend beyond the current prefix $T[1..h]$. It can be seen that in such a case the next time we need to deal with the same segment, at most one additional copy of w fits inside $T[1..h]$. This happens because the segment is relevant when $2^\ell \mid h + 2 - 2c$, and $|w| \geq 2^\ell$. Therefore, the number of times w repeats to the right can be maintained in $\mathcal{O}(1)$ time.

Theorem 5.7. *Any scheduling scheme with $b_\lambda \geq 12$ for all $\lambda \leq L$ can be simulated using $\mathcal{O}(\log n)$ additional space on the top of the space taken by the landmarks and $\mathcal{O}(\log n)$ time to process $T[h]$.*

6 Lower bounds

In this section we use Yao's minimax principle [19] to prove lower bounds on the space complexity of computing the largest radius of a palindrome in a word of length n over an alphabet Σ in the streaming model. We

denote this problem by $\text{PALIN}_\Sigma[n]$.

Theorem 6.1 (Yao's minimax principle for randomized algorithms). *Let \mathcal{X} be the set of inputs for a problem and \mathcal{A} be the set of all deterministic algorithms solving it. Then, for any $x \in \mathcal{X}$ and $A \in \mathcal{A}$, the cost of running A on x is denoted by $c(A, x) \geq 0$.*

Let p be the probability distribution over \mathcal{A} , and let A be an algorithm chosen at random according to p . Let q be the probability distribution over \mathcal{X} , and let X be an input chosen at random according to q . Then the worst-case expected cost of the randomized algorithm is at least as large as the cost of the best deterministic algorithm against the chosen distribution on the inputs:

$$\max_{x \in \mathcal{X}} \mathbf{E}[c(A, x)] \geq \min_{a \in \mathcal{A}} \mathbf{E}[c(a, X)].$$

We use the above theorem for both Las Vegas and Monte Carlo algorithms. For Las Vegas algorithms, we consider only correct algorithms, and $c(x, a)$ is the memory usage. For Monte Carlo algorithms, we consider all algorithms (not necessarily correct) with memory usage not exceeding a certain threshold, and $c(x, a)$ is the correctness indicator function, i.e., $c(x, a) = 0$ if the algorithm is correct and $c(x, a) = 1$ otherwise.

Our proofs will be based on appropriately chosen padding. The padding requires a constant number of fresh characters. If Σ is twice as large as the number of required fresh characters, we can still use half of it to construct a difficult input instance, which does not affect the asymptotics. Otherwise, we construct a difficult input instance over Σ , then add enough new fresh characters to facilitate the padding, and finally reduce the resulting larger alphabet to binary at the expense of increasing the size of the input by a constant factor.

Lemma 6.2. *For any alphabet $\Sigma = \{1, 2, \dots, \sigma\}$ there exists a morphism $h : \Sigma^* \rightarrow \{0, 1\}^*$ such that, for any $c \in \Sigma$, $|h(c)| = 2\sigma + 6$ and, for any word w , w contains a palindrome of length ℓ if and only if $h(w)$ contains a palindrome of length $(2\sigma + 6) \cdot \ell$.*

Proof. We set:

$$h(c) = 11^s 01^{s-c} 10011^{s-c} 01^c 1.$$

Clearly $|h(c)| = 2\sigma + 6$ and, because every $h(c)$ is a palindrome, if w contains a palindrome of length ℓ then $h(w)$ contains a palindrome of length $(2\sigma + 6) \cdot \ell$. Now assume that $h(w)$ contains a palindrome of length $(2\sigma + 6) \cdot \ell$, where $\ell \geq 1$. If $\ell = 1$ then we obtain that w should contain a palindrome of length 1, which always holds. Otherwise, the palindrome contains 00 inside and we consider two cases.

1. The palindrome is centered inside 00. Then it corresponds to an odd palindrome of length ℓ in w .
2. The palindrome maps some 00 to another 00. Then it corresponds to an even palindrome of length ℓ in w .

In either case, the claim holds. \square

For the padding we will often use an infinite word $\nu = 0^1 1^1 0^2 1^2 0^3 1^3 \dots$, or more precisely its prefixes of length d , denoted $\nu(d)$. Here 0 and 1 should be understood as two characters not belonging to the original alphabet, which is then reduced using the above lemma. The longest palindrome inside $\nu(d)$ has radius $\mathcal{O}(\sqrt{d})$.

We first show that any Las Vegas approximation algorithm must necessarily use $\Omega(n \log |\Sigma|)$ bits of memory in expectation in both variants, so Las Vegas randomization is essentially useless here. By Yao's minimax principle, it is enough to construct a distribution over the inputs, which is hard for any deterministic algorithm using less memory. We restrict the inputs to a family of strings of the form $\nu(E)x$$$x^R\nu(E)^R$, where $\$$ is a special character not belonging to Σ and $\nu(E)$ is a padding word of length E chosen so that there are no long palindromes inside. Then the longest palindrome must be centered in the middle of the whole word. By a counting argument, the state of the algorithm after having seen $\nu(E)x\$$ must be distinct for different words x , so the required number of bits is $\Omega(n \log |\Sigma|)$ in expectation. A bound on multiplicative approximation follows because multiplicative approximation implies additive approximation.

Theorem 6.3 (Las Vegas approximation). *Let \mathcal{A} be a Las Vegas streaming algorithm solving $\text{PALIN}_\Sigma[n]$ with additive error $E \leq 0.49n$ or multiplicative error $(1 + \varepsilon) \leq 50$ using $s(n)$ bits of memory. Then $\mathbb{E}[s(n)] = \Omega(n \log |\Sigma|)$.*

Proof. By Theorem 6.1, it is enough to construct a probability distribution \mathcal{P} over Σ^n such that for any deterministic algorithm \mathcal{D} , its expected memory usage on a word chosen according to \mathcal{P} is $\Omega(n \log |\Sigma|)$ in bits.

Consider solving $\text{PALIN}_\Sigma[n]$ with additive error E . We define \mathcal{P} as the uniform distribution over $\nu(E)x\$y\nu(E)^R$, where $x, y \in \Sigma^{n'}$, $n' = \frac{n}{2} - E - 1$, and $\$$ are special characters not belonging to Σ . Let us look at the memory usage of \mathcal{D} after having read $\nu(E)x$. We say that x is "good" when the memory usage is at most $\frac{n'}{2} \log |\Sigma|$ and "bad" otherwise. Assume that $\frac{1}{2}|\Sigma|^{n'}$ of all x 's are good, then there are two strings $x \neq x'$ such that the state of \mathcal{D} after having read both $\nu(E)x$ and $\nu(E)x'$ is exactly the same. Hence the behavior of \mathcal{D} on $\nu(E)x\$x^R\nu(E)^R$ and $\nu(E)x'\$x^R\nu(E)^R$ is exactly the same. The former is a palindrome of radius $\frac{n}{2} = n' + E + 1$, so \mathcal{D} must answer at least $n' + 1$, and consequently the latter also must contain a palindrome of radius at least $n' + 1$. A palindrome inside $\nu(E)x'\$x^R\nu(E)^R$ is either fully contained within $\nu(E)$, x' , x^R or it is a middle palindrome. But the longest palindrome inside $\nu(E)$ is of length $\mathcal{O}(\sqrt{E}) < n' + 1$ (for n large enough) and the longest palindrome inside x or x^R is of length $n' < n' + 1$, so $\nu(E)x'\$x^R\nu(E)^R$ contains a middle palindrome of radius $n' + 1$. This implies that $x = x'$, which is a contradiction. Therefore, at least $\frac{1}{2}|\Sigma|^{n'}$ of all x 's are bad. But then the expected memory usage of \mathcal{D} is at least $\frac{n'}{4} \log |\Sigma|$, which for $E \leq 0.49n$ is $\Omega(n \log |\Sigma|)$ as claimed.

Now consider solving $\text{PALIN}_\Sigma[n]$ with multiplicative error $(1 + \varepsilon)$. An algorithm with multiplicative error $(1 + \varepsilon)$ can also be considered as having additive error $E = \frac{n}{2} \cdot \frac{\varepsilon}{1 + \varepsilon}$, so if the expected memory usage of such an algorithm is $o(n \log |\Sigma|)$ and $(1 + \varepsilon) \leq 50$ then we obtain an algorithm with additive error $E \leq \frac{n}{2} \frac{49}{50} = 0.49n$ and expected memory usage $o(n \log |\Sigma|)$, which we already know to be impossible. \square

Now we move to Monte Carlo algorithms. We first consider exact algorithms solving $\text{PALIN}_\Sigma[n]$; lower bounds on approximation algorithms will be then obtained by padding the input appropriately. We introduce an auxiliary problem $\text{MID-PALIN}_\Sigma[n]$, which is to compute radius of the middle palindrome in a word of length n over an alphabet Σ . We want to show that solving $\text{MID-PALIN}_\Sigma[n]$ exactly with error probability smaller than $\frac{1}{n|\Sigma|}$ requires $\lfloor \frac{n}{2} \log |\Sigma| \rfloor$ bits of space. By Yao's minimax principle, it is enough to construct a distribution over the inputs, such that any deterministic algorithm using less memory is not able to distinguish between inputs with different answers reasonably often. This can be done by considering uniform distribution on inputs of the form $x[1] \dots x[\frac{n}{2}]x[\frac{n}{2}] \dots x[k+1]cx[k-1] \dots x[1]$. Then amplification (running multiple instances of an algorithm in parallel) gives us a lower bound on the space complexity of any algorithm solving $\text{MID-PALIN}_\Sigma[n]$ exactly. The lower bound can be translated to $\text{PALIN}_\Sigma[n]$ by padding the input in the middle, so that the longest palindrome must be centered in the middle.

Lemma 6.4. *There exists a constant γ such that any randomized Monte Carlo streaming algorithm \mathcal{A} solving $\text{MID-PALIN}_\Sigma[n]$ or $\text{PALIN}_\Sigma[n]$ exactly with probability $1 - \frac{1}{n}$ uses at least $\gamma \cdot n \log \min\{|\Sigma|, n\}$ bits of memory.*

Proof. First we prove that if \mathcal{A} is a Monte Carlo streaming algorithm solving $\text{MID-PALIN}_\Sigma[n]$ exactly using less than $\lfloor \frac{n}{2} \log |\Sigma| \rfloor$ bits of memory, then its error probability is at least $\frac{1}{n|\Sigma|}$.

By Theorem 6.1, it is enough to construct probability distribution \mathcal{P} over Σ^n such that for any deterministic algorithm \mathcal{D} using less than $\lfloor \frac{n}{2} \log |\Sigma| \rfloor$ bits of memory, the expected probability of error on a word chosen according to \mathcal{P} is at least $\frac{1}{n|\Sigma|}$.

Let $n' = \frac{n}{2}$. For any $x \in \Sigma^{n'}$, $k \in \{1, 2, \dots, n'\}$ and $c \in \Sigma$ we define:

$$w(x, k, c) = x[1]x[2]x[3] \dots x[n']x[n']x[n'-1]x[n'-2] \dots x[k+1]cx[k-1] \dots x[2]x[1].$$

Now \mathcal{P} is the uniform distribution over all such $w(x, k, c)$.

Since there are $|\Sigma|^{n'} = 2^{n' \log |\Sigma|} \geq 2 \cdot 2^{\lfloor \frac{n}{2} \log |\Sigma| \rfloor - 1}$ possible strings of length n' and we assume that \mathcal{D} uses at most $\lfloor \frac{n}{2} \log |\Sigma| \rfloor$ bits, we can partition at least half of these strings into pairs (x, x') , such that \mathcal{D} is in the same state after reading either x or x' . (If we choose an arbitrary maximal matching of strings into pairs, at

most half of possible strings will be left unpaired, that is one per each possible state of \mathcal{D} .) Let s be longest common suffix of x and x' , so $x = vcs$ and $x' = v'c's$, where $c \neq c'$ are single characters. Then \mathcal{D} returns the same answer on $w(x, n' - |s|, c)$ and $w(x', n' - |s|, c)$, even though the radius of the middle palindrome is exactly $|s|$ in one of them, and at least $|s| + 1$ in the other one. Therefore, \mathcal{D} errs on at least one of these two inputs. Similarly, it errs on either $w(x, n' - |s|, c')$ or $w(x', n' - |s|, c')$. Thus the error probability is at least $\frac{1}{2n'|\Sigma|} = \frac{1}{n|\Sigma|}$.

Now we can prove the lemma for **MID-PALIN** $_{\Sigma}[n]$ with a standard amplification trick. Say that we have a Monte Carlo streaming algorithm, which solves **MID-PALIN** $_{\Sigma}[n]$ exactly with error probability ε using $s(n)$ bits of memory. Then we can run its k instances simultaneously and return the most frequently reported answer. The new algorithm needs $\mathcal{O}(k \cdot s(n))$ bits of memory and its error probability ε_k satisfies:

$$\varepsilon_k \leq \sum_{2i < k} \binom{k}{i} (1 - \varepsilon)^i \varepsilon^{k-i} \leq 2^k \cdot \varepsilon^{k/2} = (4\varepsilon)^{k/2}.$$

Let us choose $\kappa = \frac{1}{6} \frac{\log(4/n)}{\log(1/(n|\Sigma|))} = \frac{1}{6} \frac{1-o(1)}{1+\log|\Sigma|/\log n} = \Theta(\frac{\log n}{\log n + \log|\Sigma|}) = \gamma \cdot \frac{1}{\log|\Sigma|} \log \min\{|\Sigma|, n\}$, for some constant γ . Now we can prove the theorem. Assume that \mathcal{A} uses less than $\kappa \cdot n \log|\Sigma| = \gamma \cdot n \log \min\{|\Sigma|, n\}$ bits of memory. Then running $\lfloor \frac{1}{2\kappa} \rfloor \geq \frac{3}{4} \frac{1}{2\kappa}$ (which holds since $\kappa < \frac{1}{6}$) instances of \mathcal{A} in parallel requires less than $\lfloor \frac{n}{2} \log|\Sigma| \rfloor$ bits of memory. But then the error probability of the new algorithm is bounded from above by:

$$\left(\frac{4}{n}\right)^{\frac{3}{16\kappa}} = \left(\frac{1}{n|\Sigma|}\right)^{\frac{18}{16}} \leq \frac{1}{n|\Sigma|}$$

which we have already shown to be impossible.

The lower bound for **MID-PALIN** $_{\Sigma}[n]$ can be translated into a lower bound for solving **PALIN** $_{\Sigma}[n]$ exactly by padding the input so that the longest palindrome is centered in the middle. Let $n' = \frac{n}{2}$ and $x = x[1]x[2] \dots x[n]$ be the input for **MID-PALIN** $_{\Sigma}[n]$. We define:

$$w(x) = x[1]x[2]x[3] \dots x[n'] \underbrace{1000 \dots 01}_n x[n'+1] \dots x[n].$$

Now if the radius of the middle palindrome in x is k , then $w(x)$ contains a palindrome of radius at least $n' + k + 1$. In the other direction, any palindrome inside $w(x)$ of radius larger than n' must be centered somewhere in the middle block consisting of only zeroes and both ones are mapped to each other, so it must be the middle palindrome. Thus, the radius of the longest palindrome inside $w(x)$ is exactly $n' + k + 1$, so we have reduced solving **MID-PALIN** $_{\Sigma}[n]$ to solving **PALIN** $_{\Sigma}[2n + 2]$. We already know that solving **MID-PALIN** $_{\Sigma}[n]$ with probability $1 - \frac{1}{n}$ requires $\gamma \cdot n \log \min\{|\Sigma|, n\}$ bits of memory, so solving **PALIN** $_{\Sigma}[2n + 2]$ with probability $1 - \frac{1}{2n+2} \geq 1 - \frac{1}{n}$ requires $\gamma \cdot n \log\{|\Sigma|, n\} \geq \gamma' \cdot (2n + 2) \log \min\{|\Sigma|, 2n + 2\}$ bits of memory. Notice that the reduction needs $\mathcal{O}(\log n)$ additional bits of memory to count up to n , but for large n this is much smaller than the lower bound if we choose $\gamma' < \frac{\gamma}{4}$. \square

To obtain a lower bound for Monte Carlo additive approximation, we observe that any algorithm solving **PALIN** $_{\Sigma}[n]$ with additive error E can be used to solve **PALIN** $_{\Sigma}[\frac{n-E}{E+1}]$ exactly by inserting E zeroes between every two characters, in the very beginning, and in the very end. However, this reduction requires $\log E \leq \log n$ additional bits of memory for counting up to E and cannot be used when the desired lower bound on the required number of bits $\Omega(\frac{n}{E} \log \min(|\Sigma|, \frac{n}{E}))$ is significantly smaller than $\log n$. Therefore, we need a separate technical lemma which implies that either additive or multiplicative approximation with error probability $\frac{1}{n}$ requires $\Omega(\log n)$ bits of space.

Lemma 6.5. *Let \mathcal{A} be any randomized Monte Carlo streaming algorithm solving **PALIN** $_{\Sigma}[n]$ with additive error at most $0.49n$ or multiplicative error at most $n^{0.49}$ and error probability $\frac{1}{n}$. Then \mathcal{A} uses $\Omega(\log n)$ bits of memory.*

Proof. By Theorem 6.1, it is enough to construct a probability distribution \mathcal{P} over Σ^n , such that for any deterministic algorithm \mathcal{D} using at most $s(n) = \mathcal{O}(\log n)$ bits of memory, the expected probability of error on a word chosen according to \mathcal{P} is $\frac{1}{2^{s(n)+2}}$.

Let $n' = s(n) + 1$. For any $x, y \in \Sigma^{n'}$, let $w(x, y) = \nu(\frac{n}{2} - n')^R x y^R \nu(\frac{n}{2} - n')^R$. Observe that if $x = y$ then $w(x, y)$ contains a palindrome of radius $\frac{n}{2}$, and otherwise the longest palindrome there has radius at most $2n' + \mathcal{O}(\sqrt{n}) = \mathcal{O}(\sqrt{n})$, thus any algorithm with additive error of at most $0.49n$ or with a multiplicative error at most $n^{0.49}$ must be able to distinguish between these two cases (for n large enough).

Let $S \subseteq \Sigma^{n'}$ be an arbitrary family of words of length n' such that $|S| = 2 \cdot 2^{s(n)}$, and let \mathcal{P} be the uniform distribution on all words of the form $w(x, y)$, where x and y are chosen uniformly and independently from S . By a counting argument, we can create at least $\frac{|S|}{4}$ pairs (x, x') of elements from S such that the state of \mathcal{D} is the same after having read $\nu(\frac{n}{2} - n')^R x$ and $\nu(\frac{n}{2} - n')^R x'$. (If we create the pairs greedily, at most one such x per state of memory can be left unpaired, so at least $|S| - 2^{s(n)} = \frac{|S|}{2}$ elements are paired.) Thus, \mathcal{D} cannot distinguish between $w(x, x')$ and $w(x, x)$, and between $w(x', x')$ and $w(x', x)$, so its error probability must be at least $\frac{|S|/2}{|S|^2} = \frac{1}{4 \cdot 2^{s(n)}}$. Thus if $s(n) = o(\log n)$, the error rate is at least $\frac{1}{n}$ for n large enough, a contradiction. \square

Combining the reduction with the technical lemma and taking into account that we are reducing to a problem with word length of $\Theta(\frac{n}{E})$, we obtain the following.

Theorem 6.6 (Monte Carlo additive approximation). *Let \mathcal{A} be any randomized Monte Carlo streaming algorithm solving $\text{PALIN}_\Sigma[n]$ with additive error E with probability $1 - \frac{1}{n}$. If $E \leq 0.49n$ then \mathcal{A} uses $\Omega(\frac{n}{E} \log \min\{|\Sigma|, \frac{n}{E}\})$ bits of memory.*

Proof. Define $\sigma = \min\{|\Sigma|, \frac{1}{2} \frac{n}{E}\}$.

Because of Lemma 6.5 and $\log \sigma \geq \frac{1}{35} \log \min\{|\Sigma|, \frac{n}{E}\}$ (which holds due to $E \leq 0.49n$), it is enough to prove that $\Omega(\frac{n}{E} \log \sigma)$ is a lower bound when

$$E \leq \frac{\gamma}{4} \cdot \frac{n}{\log n} \log \sigma. \quad (5)$$

Assume that there is a Monte Carlo streaming algorithm \mathcal{A} solving $\text{PALIN}_\Sigma[n]$ with additive error E using $o(\frac{n}{E} \log \sigma)$ bits of memory and probability $1 - \frac{1}{n}$. Let $n' = \frac{n-E}{E+1} \geq \frac{1}{2} \frac{n}{E}$ (the last inequality holds because $E \leq 0.49n$ and because we can assume that $E > 1$). Given a word $x[1]x[2] \dots x[n']$, we can simulate running \mathcal{A} on $0^E x[1]0^E x[2]0^E x[3] \dots 0^E x[n']0^E$ to calculate R (using $\log E \leq \log n$ additional bits of memory), and then return $\lfloor \frac{R}{E+1} \rfloor$. We call this new Monte Carlo streaming algorithm \mathcal{A}' . Recall that \mathcal{A} reports the radius of the longest palindrome with additive error E . Therefore, if the original word contains a palindrome of radius r , the new word contains a palindrome of radius $\frac{E}{2} + r(E+1)$, so $R \geq r(E+1)$ and \mathcal{A}' will return at least r . In the other direction, if \mathcal{A}' returns r , then the new word contains a palindrome of radius $r(E+1)$. If such palindrome is centered so that $x[i]$ is matched with $x[i+1]$ for some i , then it clearly corresponds to a palindrome of radius r in the original word. But otherwise every $x[i]$ within the palindrome is matched with 0, so in fact the whole palindrome corresponds to a streak of consecutive zeroes in the new word and can be extended to the left and to the right to start and end with 0^E , so again it corresponds to a palindrome of radius r in the original word. Therefore, \mathcal{A}' solves $\text{PALIN}_\Sigma[n']$ exactly with probability $1 - \frac{1}{(n'(E+1)+E)} \geq 1 - \frac{1}{n'}$ and uses $o(\frac{n'(E+1)+E}{E} \log \sigma) + \log n = o(n' \log \sigma) + \log n$ bits of memory. Observe that by Lemma 6.4 we get a lower bound

$$\gamma \cdot n' \log \min\{|\Sigma|, n'\} \geq \frac{\gamma}{2} \cdot n' \log \sigma + \frac{\gamma}{4} \cdot \frac{n}{E} \log \sigma \geq \frac{\gamma}{2} \cdot n' \log \sigma + \log n$$

(where the last inequality holds because of Eq.(5)). Then, for large n we obtain contradiction as follows

$$o(n' \log \sigma) + \log n < \frac{\gamma}{2} \cdot n' \log \sigma + \log n. \quad \square$$

Finally, we consider multiplicative approximation. Here we observe that any algorithm solving **PALIN** $_{\Sigma}[n]$ with multiplicative error $(1 + \varepsilon)$ can be used to solve **MID-PALIN** $_{\Sigma}[2n']$ exactly, where $n' = \Theta(\frac{\log n}{\log(1+2\varepsilon)})$, by separating the characters appropriately. Intuitively, the padding is chosen so that the middle palindrome has the largest radius and the larger the distance from the center the longer the separator inserted between two consecutive characters of the original input. Again, we need $\log n$ bits for a counter and hence need to invoke a separate technical lemma when $(1 + \varepsilon)$ is very large. After some calculations, and taking into account that we are reducing to a problem with word length of $\Theta(\frac{\log n}{\log(1+\varepsilon)})$ we obtain the following.

Theorem 6.7 (Monte Carlo multiplicative approximation). *Let \mathcal{A} be any randomized Monte Carlo streaming algorithm solving **PALIN** $_{\Sigma}[n]$ with multiplicative error $(1 + \varepsilon)$ with probability $1 - \frac{1}{n}$. If $n^{-0.98} \leq \varepsilon \leq n^{0.49}$ then \mathcal{A} uses $\Omega(\frac{\log n}{\log(1+\varepsilon)} \log \min\{|\Sigma|, \frac{\log n}{\log(1+\varepsilon)}\})$ bits of memory.*

Proof. For $\varepsilon \geq n^{0.001}$ then the claimed lower bound reduces to $\Omega(1)$ bits, which obviously holds. Thus we can assume that $\varepsilon < n^{0.001}$. Define

$$\sigma = \min\{|\Sigma|, \frac{1}{50} \frac{\log n}{\log(1+2\varepsilon)} - 2\}.$$

First we argue that it is enough to prove that \mathcal{A} uses $\Omega(\frac{\log n}{\log(1+\varepsilon)} \log \sigma)$ bits of memory. Since $\log(1+2\varepsilon) \leq 0.001 \log n + o(\log n)$, we have that:

$$\frac{1}{50} \frac{\log n}{\log(1+2\varepsilon)} - 2 \geq 18 - o(1) \quad (6)$$

and consequently:

$$\frac{1}{50} \frac{\log n}{\log(1+2\varepsilon)} - 2 = \Theta(\frac{\log n}{\log(1+2\varepsilon)}). \quad (7)$$

Finally, observe that:

$$\log(1+2\varepsilon) = \Theta(\log(1+\varepsilon)) \quad (8)$$

because $\log 2(1+\varepsilon) = \Theta(\log(1+\varepsilon))$ for $\varepsilon \geq 1$, and $\log(1+\varepsilon) = \Theta(\varepsilon)$ for $\varepsilon < 1$. From (7) and (8) we conclude that:

$$\log \sigma = \Theta(\log \min\{|\Sigma|, \frac{\log n}{\log(1+\varepsilon)}\}). \quad (9)$$

Because of Lemma 6.5 and equations (8) and (9), it is enough to prove that $\Omega(\frac{\log n}{\log(1+\varepsilon)} \log \sigma)$ is a lower bound when

$$\log(1+2\varepsilon) \leq \gamma \cdot \frac{\log \sigma}{100}, \quad (10)$$

as otherwise $\Omega(\frac{\log n}{\log(1+\varepsilon)} \log \sigma) = \Omega(\frac{\log n}{\log(1+2\varepsilon)} \log \sigma) = \Omega(\log n)$.

Assume that there is a Monte Carlo streaming algorithm \mathcal{A} solving **PALIN** $_{\Sigma}[n]$ with multiplicative error $(1+\varepsilon)$ with probability $1 - \frac{1}{n}$ using $o(\frac{\log n}{\log(1+\varepsilon)} \log \sigma)$ bits of memory. Let $x = x[1]x[2] \dots x[n']x[n'+1] \dots x[2n']$ be an input for **MID-PALIN** $_{\Sigma}[2n']$. We choose n' so that $n = (1+2\varepsilon)^{n'+1} \cdot n^{0.99}$. Then $n' = \log_{(1+2\varepsilon)}(n^{0.01}) - 1 = \frac{1}{100} \frac{\log n}{\log(1+2\varepsilon)} - 1$. We choose $i_0, i_1, i_2, i_3, \dots, i_{n'}$ so that $i_0 + \dots + i_d = \lceil (1+2\varepsilon)^{d+1} \cdot n^{0.99} \rceil$ for any $0 \leq d \leq n'$.

(Observe that for $\varepsilon = \Omega(n^{-0.98})$ we have $i_0 > n^{0.99}$ and $i_1, \dots, i_d > 2n^{0.01} - 1$.) Finally we define:

$$w(x) = \nu(i_{n'})^R x[1] \nu(i_{n'-1})^R \dots x[n'] \nu(i_0)^R \nu(i_0) x[n'+1] \nu(i_1) \dots \nu(i_{n'-1}) x[2n'] \nu(i_{n'}).$$

If x contains a middle palindrome of radius exactly k , then $w(x)$ contains a middle palindrome of radius $(1+2\varepsilon)^{k+1} \cdot n^{0.99}$. Also, based on the properties of ν , any non-middle centered palindrome in $w(x)$ has radius at most $\mathcal{O}(\sqrt{n})$, which is less than $n^{0.99}$ for n large enough. Since $\lceil (1+2\varepsilon)^k \cdot n^{0.99} \rceil \cdot (1+\varepsilon) < ((1+2\varepsilon)^k \cdot n^{0.99} + 1) \cdot (1+\varepsilon) < (1+2\varepsilon)^{k+1} \cdot n^{0.99}$, value of k can be extracted from the answer of \mathcal{A} . Thus, if \mathcal{A} approximates the middle palindrome in $w(x)$ with multiplicative error $(1+\varepsilon)$ with probability $1 - \frac{1}{n}$ using

$o(\frac{\log n}{\log(1+\varepsilon)} \log \sigma)$ bits of memory, we can construct a new algorithm \mathcal{A}' solving **MID-PALIN** $_{\Sigma}[2n']$ exactly with probability $1 - \frac{1}{n} > 1 - \frac{1}{2n'}$ using

$$o(\frac{\log n}{\log(1+\varepsilon)} \log \sigma) + \log n \quad (11)$$

bits of memory. By Lemma 6.4 we get a lower bound

$$\begin{aligned} \gamma \cdot 2n' \log \min\{|\Sigma|, 2n'\} &= \frac{\gamma}{50} \cdot \frac{\log n}{\log(1+2\varepsilon)} \log \sigma - 2\gamma \log \sigma \\ &\geq \frac{\gamma}{100} \cdot \frac{\log n}{\log(1+2\varepsilon)} \log \sigma + \log n - 2\gamma \log \sigma \end{aligned} \quad (12)$$

(where the last inequality holds because of (10)). On the other hand, for large n

$$\begin{aligned} \frac{\gamma}{100} \cdot \frac{\log n}{\log(1+2\varepsilon)} \log \sigma - 2\gamma \log \sigma + \log n &= \left(\frac{1}{100} \frac{\log n}{\log(1+2\varepsilon)} - 2 \right) \gamma \log \sigma + \log n \\ &= \Theta \left(\frac{\log n}{\log(1+\varepsilon)} \log \sigma \right) + \log n \end{aligned}$$

so (12) exceeds (11), a contradiction. □

Acknowledgments

The first author is currently holding a post-doctoral position at Warsaw Center of Mathematics and Computer Science. However, most of this work has been done when the first author was at Max-Planck-Institut für Informatik and the second author at LIF, CNRS – Aix Marseille University (supported by the Labex Archimède and by the ANR project MACARON (ANR-13-JS02-0002)).

The authors would like to thank Tomasz Syposz for a suggestion which allowed them to simplify the algorithm.

References

- [1] Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. *Theor. Comput. Sci.*, 141(1&2):163–173, 1995.
- [2] Petra Berenbrink, Funda Ergün, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer. Palindrome Recognition In The Streaming Model. In *STACS 2014*, volume 25 of *LIPICs*, pages 149–161, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [3] Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Transactions on Algorithms*, 10(4):22, 2014.
- [4] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2015.
- [5] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k -mismatch problem revisited. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 2039–2052. SIAM, 2016.
- [6] Funda Ergün, Hossein Jowhari, and Mert Saglam. Periodicity in streams. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, volume 6302 of *Lecture Notes in Computer Science*, pages 545–559. Springer, 2010.
- [7] Gabriele Fici, Travis Gagie, Juha Kärkkäinen, and Dominik Kempa. A subquadratic algorithm for minimum palindromic factorization. *J. Discrete Algorithms*, 28:41–48, 2014.
- [8] N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the AMS*, 16:109–114, 1965.
- [9] Zvi Galil and Joel Seiferas. A linear-time on-line recognition algorithm for “palstar”. *J. ACM*, 25(1):102–111, January 1978.
- [10] Paweł Gawrychowski, Florin Manea, and Dirk Nowotka. Testing Generalised Freeness of Words. In *STACS 2014*, volume 25 of *LIPICs*, pages 337–349, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [11] Tomohiro I, Shiho Sugimoto, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing palindromic factorizations and palindromic covers on-line. In *CPM 2014*, volume 8486 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2014.

- [12] Markus Jalsenius, Benny Porat, and Benjamin Sach. Parameterized matching in the streaming model. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 - March 2, 2013, Kiel, Germany*, pages 400–411, 2013.
- [13] Haim Kaplan and Robert E. Tarjan. Persistent lists with catenation via recursive slow-down. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing, STOC '95*, pages 93–102, New York, NY, USA, 1995. ACM.
- [14] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.
- [15] Donald E. Knuth, Jr. James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.
- [16] Dmitry Kosolobov, Mikhail Rubinchik, and Arseny M. Shur. Pal^k is linear recognizable online. In *SOFSEM 2015*, volume 8939 of *Lecture Notes in Computer Science*, pages 289–301. Springer, 2015.
- [17] Glenn K. Manacher. A new linear-time “on-line” algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, 1975.
- [18] Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 315–323. IEEE Computer Society, 2009.
- [19] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *FOCS*, pages 222–227. IEEE Computer Society, 1977.